RAPPORT DE PROJET : SNAKE

Moncef STITI & Marco ORFAO

SOMMAIRE

INTRODUCTION

Dans le cadre de la SAé S1.01, il nous as été demandé de concevoir une version du jeu du serpent : SNAKE 🐍.
Le jeu du Snake consiste à guider un serpent constamment en mouvement à travers un terrain constitué d'une grille de 40 lignes par 60 colonnes. L'objectif du joueur est de contrôler le serpent de manière à le faire manger des pommes dispersées aléatoirement sur le terrain, tout en évitant de heurter les limites du terrain ou sa propre queue. Le serpent se déplace continuellement, ajoutant un segment (de la taille d'une case) en tête et retirant un segment en queue à chaque déplacement. Le projet implique la programmation en langage C89, en utilisant uniquement la bibliothèque graphique de l'IUT pour mener à bien notre projet.

Notre professeur nous as donné quelques contraintes/règles obligatoires telles que :

- La gestion des déplacements du serpent à l'aide des touches directionnelles
- La possibilité de quitter la partie avec la touche Échap
- La possibilité de mettre en pause la partie avec la touche Espace
- La visualisation du temps écoulé depuis le début de la partie
- La visualisation du score de l'utilisateur (égal à 5 fois le nombre de pommes consommées)

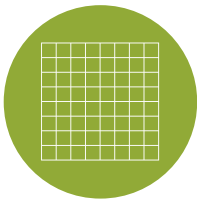
Nous avions également le choix entre plusieurs variantes à implémenter dans notre jeu telles que :

- Le terrain de jeu comporte des obstacles à éviter
- Le déplacement du serpent s'accélère en fonction du score ou du temps

...

BRAINSTORMING

Avant de commencer à coder, nous avons fait un brainstorming afin de définir tout les fichiers et fonction à créer ainsi que les fonctionnalités que nous voulions implémenter à notre jeu.



N° 01 - GRILLE.C

Premièrement, nous avons besoin d'une grille bicolore de 40 lignes par 60 colonnes pour améliorer l'expérience de jeu de l'utilisateur.

N° 02 - SERPENT.C

Nous avons également besoin d'un serpent capable de grandir, se déplacer et de mourrir s'il sort de la grille ou se mange la queue.

N° 03 - POMME.C

Le serpent est affamé, il nous donc de quoi le rassasier ! Nous avons donc besoin d'afficher des pommes à l'aide d'un sprite et de vérifier si le serpent à manger une pomme.

N° 04 - MENU.C

Comment pourrions-nous appeler cela un jeu si nous n'avions pas de menus... Il nous faut donc un menu principal, un menu modes de jeux et bien évidemment un menu game over.

N° 05 - TIMER.C

Comme tout jeux addictif, il nous faut un moyen de battre ses amis ! Il nous faut donc un timer pour permettre à l'utilisateur de battre les records de temps de ses amis.

N° 06 - JEU.C

Maintenant que nous avons tout nos programmes, il nous faut les rassembler pour donner vie à nos différents modes de jeux.

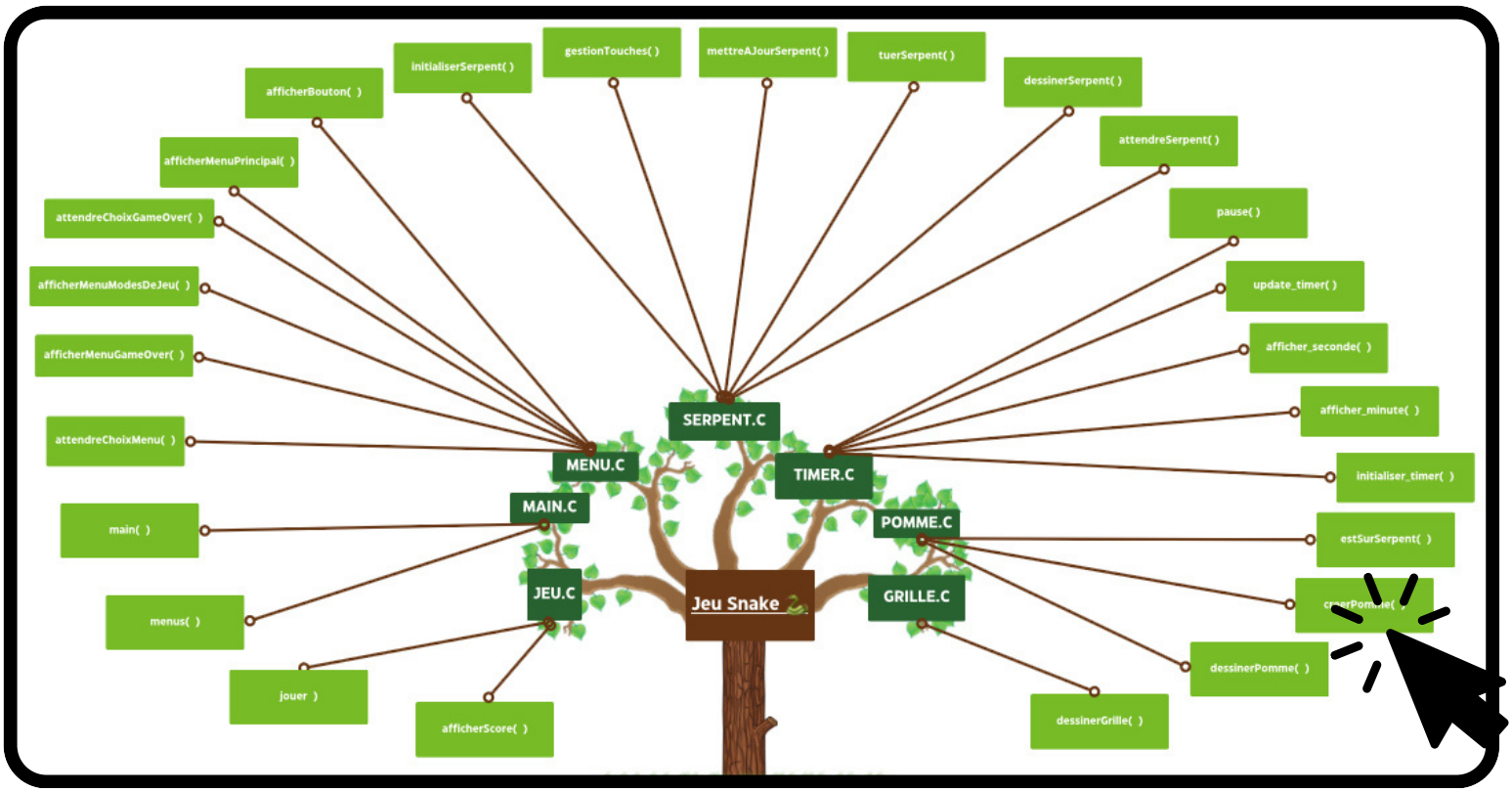
N° 07 - MAIN.C

Pour finir, nous avons besoin d'un programme capable de lancer nos menus, nos modes de jeux etc... sans que l'utilisateur ai à taper des lignes de code pour lancer son mode préféré 😅

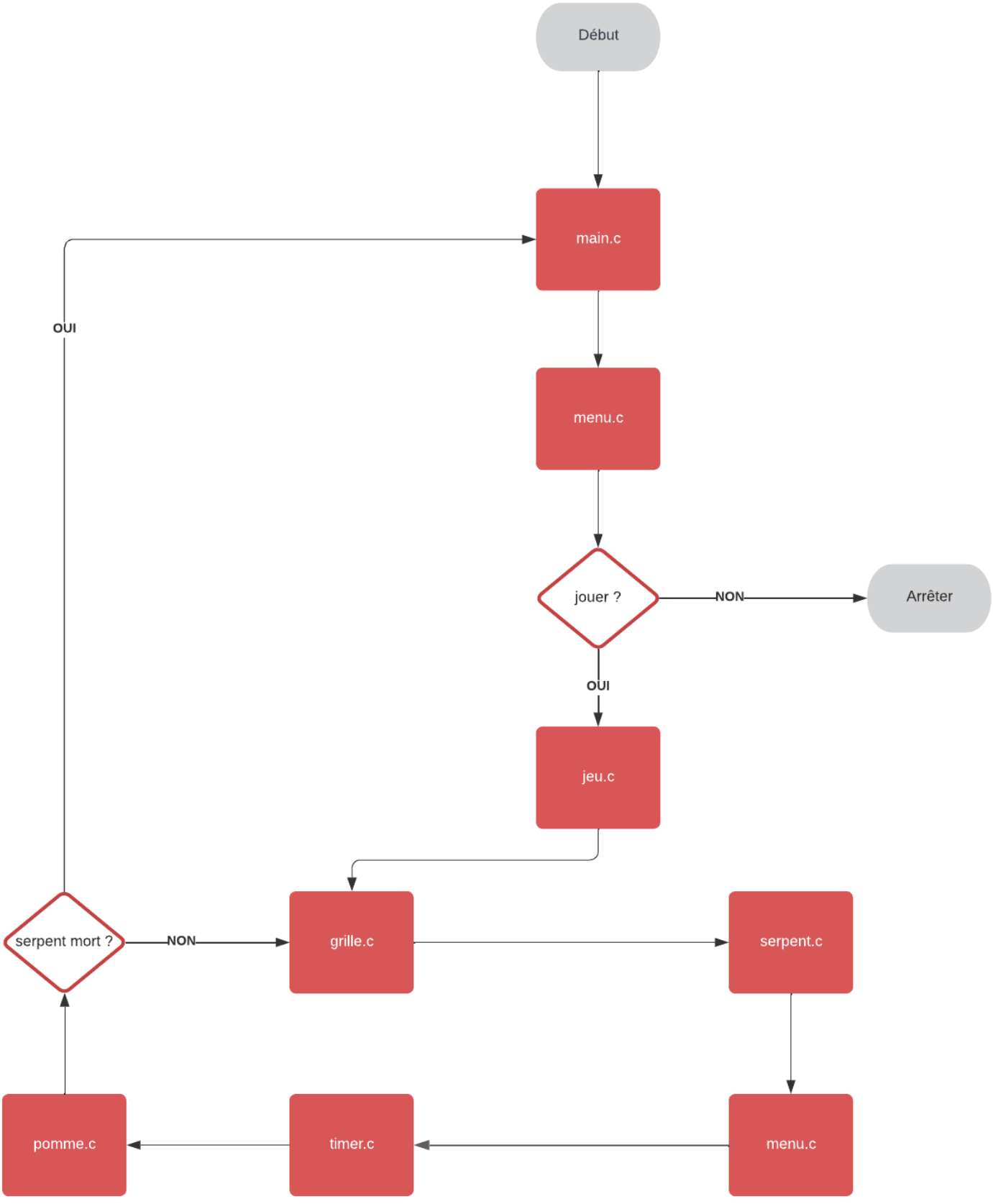
LE SCHÉMA POST BRAINSTORMING

Nous avons planifier l'ensemble des fonctions dont nous aurions besoins pour ce projet et avons crée un schéma nous permettant de visualiser les tâches à accomplir.

Voici le schéma :



Voici l’algorigramme avec les différents fichiers de notre projet :



DESCRIPTION DES FICHIERS

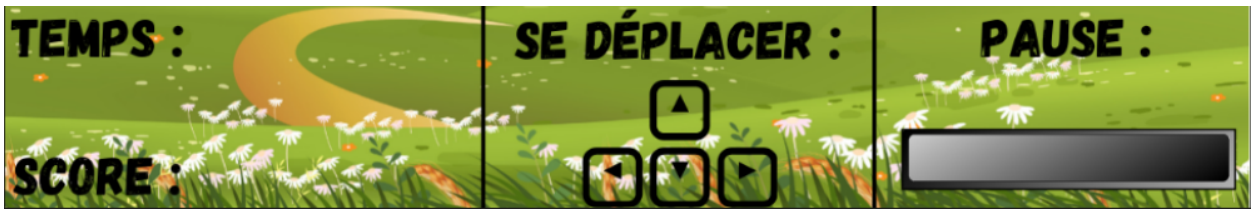
GRILLE.C

Le fichier grille.c ne contient qu'une fonction se nommant "dessinerGrille".
Cette fonction permet d'afficher graphiquement une grille de 60 colonnes pour 40 lignes.
Chaque cases de la grille alterne entre une couleur vert foncé et vert clair pour apporter un meilleur confort visuel au joueurs.

Voici à quoi ressemble la grille :

dessinerGrille affiche aussi une image en dessous de la grille recensant des informations tel que : le score, le temps écoulé depuis le début de la partie, les touches pour contrôler le serpent ainsi que pour mettre en pause la partie.

Voici à quoi ressemble l'image :



La fonction initialiserSerpent

- **But** : Cette fonction initialise le serpent au centre de la grille en lui donnant une longueur de 10 segments (= 10 cases de la grille). Les segments sont représentés par une structure qui contient les coordonnées x et y. Cette fonction permet à notre programme de se réinitialiser à chaque partie.
- **Paramètres** : Cette fonction prend 2 paramètres :

  - **Serpent serpent :** La structure serpent (c'est un alias de segment) qu permet de stocker les positions x et y.
  - **int* longueur** : Un pointeur vers une variable de type int qui représente la longueur du serpent

La fonction gestionTouches

- **But** : Cette fonction gère les touches du clavier pour modifier la direction du serpent. Elle détecte les touches de déplacement (haut, bas, gauche, droite), la barre d'espace pour la pause, et la touche Escape pour quitter.
- **Paramètres** : Cette fonction prend 3 paramètres :

  - **Serpent serpent** : La structure serpent qui représente le serpent.
  - **int* direction_x** : Un pointeur vers la direction horizontale du serpent.
  - **int* direction_y** : Un pointeur vers la direction verticale du serpen
- **Retour** : La fonction retourne 1 si la touche Escape est pressée, sinon 0.

La fonction tuerSerpent

- **But** : Cette fonction vérifie si le serpent est mort en sortant de la grille ou en entrant en collision avec son propre corps.
- **Paramètres** : Cette fonction prend 2 paramètres :

  - **Serpent serpent :** La structure serpent (c'est un alias de segment) qu permet de stocker les positions x et y.

  - **int* longueur** : Un pointeur vers une variable de type int qui représente la longueur du serpent
- **Retour** : La fonction retourne 1 si le serpent est mort (en entrant en contacte avec son propre corps ou en sortant des limites du terrain), sinon 0.

La fonction mettreAJourSerpent

- **But** : Cette fonction est extrêmement importante car elle effectue la mise à jour de la position du serpent. Elle déplace le serpent d'une case dans la direction spécifiée par les variables direction_x et direction_y, met à jour visuellement la grille en effaçant la dernière position du serpent et en dessinant la nouvelle position, et vérifie également si le serpent a rencontré des conditions conduisant à sa mort avec tuerSerpent.

  ATTENTION : Cette fonction ne dessine pas le serpent, elle actualise simplement les informations nécessaires pour les fonctions de dessin !
- **Paramètres** : Cette fonction prend 4 paramètres :

  - **Serpent serpent** : La structure serpent qui représente le serpent.

  - **int* longueur** : Un pointeur vers une variable de type int qui représente la longueur du serpent

  - **int* direction_x** : Un pointeur vers la direction horizontale du serpent.

  - **int* direction_y** : Un pointeur vers la direction verticale du serpent

- **Fonctionnement détaillé** :

1. **Sauvegarde des anciennes coordonnées de la queue du serpent :** Les coordonnées de la dernière case du serpent (**ancienX** et **ancienY**) sont sauvegardées, car c'est la position qui sera effacée après le déplacement.
2. **Détermination de la couleur de fond :** La couleur de fond pour la nouvelle case est déterminée en fonction des coordonnées de la dernière case du serpent. Cela permet d'alterner les couleurs de fond dans la grille.
3. **Effacement de la dernière position du serpent :** La dernière position du serpent est effacée en utilisant la couleur de fond appropriée.
4. **Déplacement du corps du serpent :** Les segments du corps du serpent sont mis à jour en déplaçant chaque segment vers la position du segment précédent, de la queue vers la tête.
5. **Déplacement de la tête du serpent :** Les coordonnées de la tête du serpent sont mises à jour en fonction de la direction spécifiée par les variables **direction_x** et **direction_y**.

La fonction attendreSerpent

- **But** : Cette fonction introduit un délai dans l'exécution du programme en attendant un nombre spécifié de microsecondes. Elle est utilisée pour contrôler la vitesse du serpent dans le jeu.
- **Paramètres** : Cette fonction prend 1 paramètre :

- **unsigned long int microseconds** : Le nombre de microsecondes à attendre.

La fonction dessinerSerpent

- **But** : Cette fonction dessine visuellement le serpent sur la grille en utilisant des rectangles colorés. La tête du serpent est représentée en jaune avec un oeil noir (pour différencier la tête de la queue), et le reste du corps est également en jaune.

- **Paramètres** : Cette fonction prend 2 paramètres :

  - **Serpent serpent** : La structure serpent qui représente le serpent.
  - **int* longueur** : Un pointeur vers une variable de type int qui représente la longueur du serpent

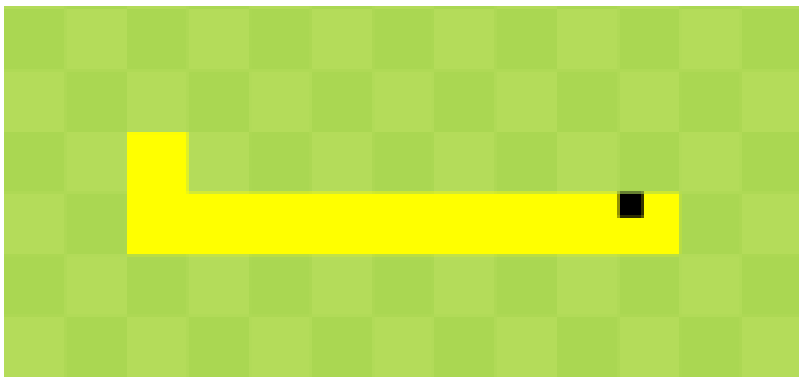
- **Fonctionnement détaillé** :

1. **Dessin de la tête du serpent** : La tête du serpent est dessinée en jaune en utilisant la fonction RemplirRectangle. De plus, les yeux noirs sont dessinés en noir pour donner une apparence distincte à la tête.
2. **Répétition pour chaque segment du corps** : La boucle for parcourt chaque segment du corps du serpent (à partir du deuxième segment, car la tête a déjà été dessinée). Pour chaque segment, un rectangle jaune est dessiné à la position correspondante.

Informations complémentaires sur le Serpent

Le Serpent, étant l'élément phare du projet, nous devons donner quelques informations complémentaires importantes sur son fonctionnement :

- Tout d'abord, le Serpent est initialisé de manière dynamique dans le but de pouvoir le libérer en fin de partie.
- L'espace alloué de 2400 fois la taille de son type représente sa taille maximale.
    - (40 lignes x 60 colonnes = 2400 cases)
- Le Serpent est redessiné à chaque tour de boucle et la case qui était précédemment positionner à la fin du serpent se recolorie à sa couleur initiale.



La fonction estSurSerpent

- **But** : Cette fonction vérifie si une pomme se trouve sur le corps du serpent en comparant les coordonnées de la pomme avec les coordonnées de chaque segment du serpent. Cette fonction est cruciale car elle permet aux pommes de ne jamais apparaitre sur le serpent.
- **Paramètres** : Cette fonction prend 3 paramètres :
  - **Pomme pomme** : La pomme à vérifier.
  - **Serpent serpent** : La structure serpent qui représente le serpent. représente la longueur du serpent.
  - **int longueur** : La longueur actuelle du serpent.
- **Retour** : La fonction retourne 1 si la pomme est sur le serpent, sinon elle retourne 0.
- **Fonctionnement détaillé** :

1. **Boucle de parcours :** Une boucle for parcourt chaque segment du serpent.
2. **Comparaison des coordonnées :** Pour chaque segment du serpent, les coordonnées de la pomme sont comparées avec celles du segment. Si elles sont égales, la fonction retourne 1 indiquant que la pomme est sur le serpent.
3. **Fin de boucle :** Si la boucle parcourt tous les segments sans trouver de correspondance, la fonction retourne 0 indiquant que la pomme n'est pas sur le serpent.

La fonction creerPomme

- **But** : Cette fonction crée une nouvelle pomme en générant aléatoirement des coordonnées qui ne se trouvent pas sur le serpent.
- **Paramètres** : Cette fonction prend 3 paramètres :

  - **Serpent serpent** : La structure serpent qui représente le serpent. représente la longueur du serpent.
  - **int longueur** : La longueur actuelle du serpent.
- **Fonctionnement détaillé** :

1. **Initialisation de la pomme :** Une nouvelle pomme est créée avec des coordonnées aléatoires.
2. **Vérification de la position :** Une boucle do-while est utilisée pour s'assurer que la pomme générée n'est pas sur le serpent. Si elle est sur le serpent, de nouvelles coordonnées sont générées jusqu'à ce que la pomme soit positionnée hors du serpent.
3. **Retour de la pomme :** Une fois que des coordonnées valides ont été trouvées, la pomme est retournée.

La fonction dessinerPomme

- **But** : Affiche la pomme sur la grille à l'emplacement spécifié par ses coordonnées.
- **Paramètres** : Cette fonction prend 2 paramètres :

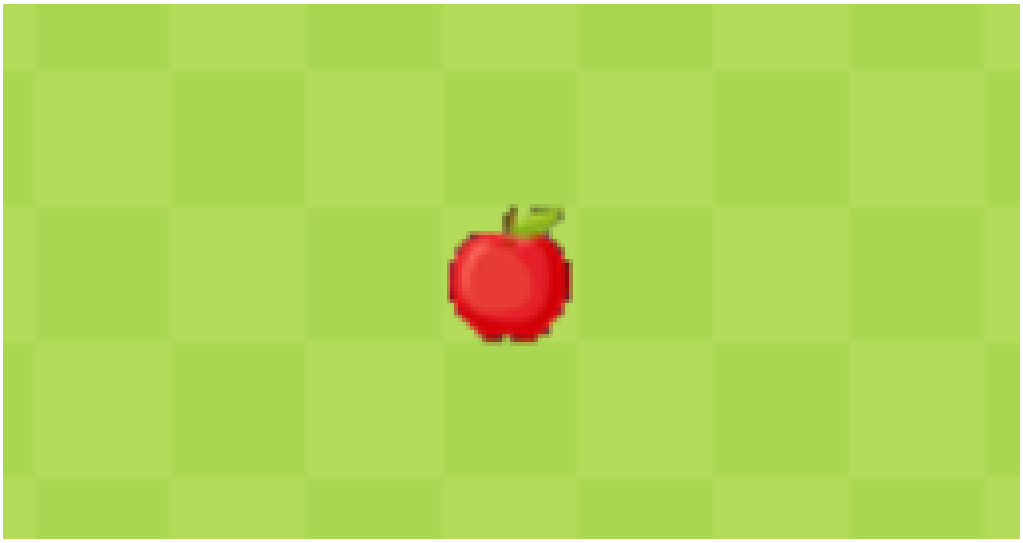
  - **Pomme pomme** : La pomme à dessiner.
  - **int id_pomme** : Identifiant du sprite de la pomme (définis dans jeu.c)

Informations complémentaires sur les pommes

Nous avons fait le choix d'utiliser un sprite pour afficher la pomme. Cette décision nous as permis d'afficher une pomme réaliste plutôt qu'un simple carré rouge.

Nous avons utilisé une pomme libre de droit d'auteur que nous avons mis au format 20px x 20px (exactement la taille d'une case).

Voici à quoi ressemble une pomme :



MENU.C

La fonction afficherMenuPrincipal

- **But** : Cette fonction affiche le menu principal avec les boutons "Jouer" et "Quitter".
- **Fonctionnement détaillé** :

1. **Chargement de l'image de fond :** L'image de fond du menu principal est chargée avec la fonction ChargerImageFond.
2. **Affichage du bouton "Jouer" :** Le bouton "Jouer" est affiché avec la fonction afficherBouton. Les coordonnées, le texte, les couleurs de fond, de bordure et de texte sont spécifiés.
3. **Affichage du bouton "Quitter" :** Le bouton "Quitter" est affiché de la même manière que le bouton "Jouer".

La fonction afficherMenuModesDeJeu

- **But** : Cette fonction affiche le menu des modes de jeu avec les boutons "Classique", "MultiPommes", "Glouton" et "Acceleration" (correspondant à nos 4 modes de jeux).
- **Fonctionnement détaillé** :

1. **Chargement de l'image de fond :** L'image de fond des modes de jeu est chargée avec la fonction ChargerImageFond.
2. **Affichage des boutons de modes de jeu :** Chaque bouton est affiché avec la fonction afficherBouton. Les coordonnées, le texte, les couleurs de fond, de bordure et de texte sont spécifiés.

La fonction afficherMenuGameOver

- **But** : Cette fonction affiche le menu “Game Over”.
- **Fonctionnement détaillé** :

1. **Chargement de l'image de fin de jeu** : L'image de fin de jeu est chargée avec la fonction ChargerImageFond.

La fonction attendreChoixGameOver

- **But** : Cette fonction attend que l'utilisateur clique sur le bouton "Menu Principal" dans le menu “Game Over”.
- **Fonctionnement détaillé** :

1. **Boucle d'attente :** Une boucle infinie vérifie constamment si la souris a été cliquée.
2. **Vérification du clic :** Si la souris est cliquée, les coordonnées de la souris sont obtenues.
3. **Vérification du clic sur "Menu Principal" :** Si les coordonnées cliquées se trouvent dans la zone du bouton "Menu Principal", la fonction retourne (ce qui nous permet de quitter la fonction).

La fonction attendreChoixModesDeJeu

- **But** : Cette fonction attend que l'utilisateur choisisse un mode de jeu en cliquant sur l'un des boutons du menu des modes de jeu.
- **Fonctionnement détaillé** :

1. **Boucle d'attente** : Une boucle infinie vérifie constamment si la souris a été cliquée.
2. **Vérification du clic** : Si la souris est cliquée, les coordonnées de la souris sont obtenues.
3. **Vérification du clic sur les boutons** : Si les coordonnées cliquées se trouvent dans la zone de l'un des boutons, la fonction renvoie le numéro du bouton cliqué.

La fonction attendreChoixMenu

- **But** : Cette fonction attend que l'utilisateur choisisse entre “Jouer” ou “Quitter” en cliquant sur l'un des boutons du menu.
- **Fonctionnement détaillé** :

1. **Boucle d'attente :** Une boucle infinie vérifie constamment si la souris a été cliquée.
2. **Vérification du clic :** Si la souris est cliquée, les coordonnées de la souris sont obtenues.
3. **Vérification du clic sur les boutons :** Si les coordonnées cliquées se trouvent dans la zone de l'un des boutons, la fonction renvoie le numéro du bouton cliqué.

La fonction afficherBouton

- **But** : Cette fonction permet de crée un bouton. Afin de ne pas répéter cette étapes pour nos 6 boutons, nous avons crée une fonction qui crée des boutons.
- **Paramètres** : Cette fonction prend 2 paramètres :

  - **double x1, double y1, double x2, double y2** : Coordonnées du coin supérieur gauche et du coin inférieur droit du bouton.
  - **const char *texte** : Texte à afficher sur le bouton.
  - **couleur arriere_plan, couleur bordure, couleur couleur_texte** : Couleurs de l'arrière-plan, de la bordure et du texte du bouton.
  - **int taille_texte** : Taille du texte du bouton.

TIMER.C

La fonction initialiser_timer

- **But** : Cette fonction permet d'initialiser les minutes ainsi que les secondes à 0. Elle est appelé en début de partie pour initialiser le Timer à 0.
- **Paramètres** : Cette fonction prend 2 paramètres :

- int *min : L'adresse de la variable contenant un entier qui représente les minutes.

- int *sec : L'adresse de la variable contenant un entier qui représente les secondes.

La fonction afficher_minute

- **But** : Cette fonction permet d'afficher graphiquement à des coordonnées x et y prédéfinies les minutes sur le Timer.
- **Paramètres** : Cette fonction prend 1 paramètres :

- int min : La variable qui stock les minutes écoulées depuis le début du jeu

La fonction afficher_seconde

- **But** : Cette fonction permet d'afficher graphiquement à des coordonnées x et y prédéfinies les secondes sur le Timer.
- **Paramètres** : Cette fonction prend 1 paramètres :

- int sec : La variable qui stock les secondes écoulées depuis le début du jeu

La fonction update_timer

- **But** : Cette fonction permet de mettre à jour les minutes et les secondes du Timer en utilisant la fonction “Microsecondes” de la bibliothèque graphique. Cette fonction va se baser sur un cycle de 1.000.000 (un million de microsecondes qui équivaut à une seconde) et va mettre à jour la variable sec à chaque cycle de un millions de microsecondes. La valeur dans la variable min prends +1 toutes les 60 secondes.
- **Paramètres** : Cette fonction prend 2 paramètres :

  - int *min : L’adresse de la variable contenant un entier qui représente les minutes.

  - int *sec : L’adresse de la variable contenant un entier qui représente les secondes.

voici le Timer : 

La fonction pause

- **But** : Cette fonction permet de mettre en pause le jeu, et plus implicitement le programme dans sa globalité grâce à une boucle infinie qui prends fin que lorsque la touche “Espace” est appuyée.

JEU.C

La fonction afficherScore

- **But** : Cette fonction permet de convertir le score en une chaîne de caractères à l'aide de snprintf afin de l'afficher graphiquement tout en effaçant l'ancien score).
- **Paramètres** : Cette fonction prend 1 paramètres :

  - int score : Le score actuelle

La fonction jouer

- **But** : Cette fonction permet de crée des modes de jeux.
  - Avec ou sans pommes
  - Choix de la vitesse
  - Avec ou sans modes accélération
- **Paramètres** : Cette fonction prend 3 paramètres :

  - int nbPommes : Le nombre de pommes souhaitées

  - **unsigned long int vitesse** : La vitesse du serpent

  - int acceleration : L'activation ou non du mode accélération

MAIN.C

La fonction menus

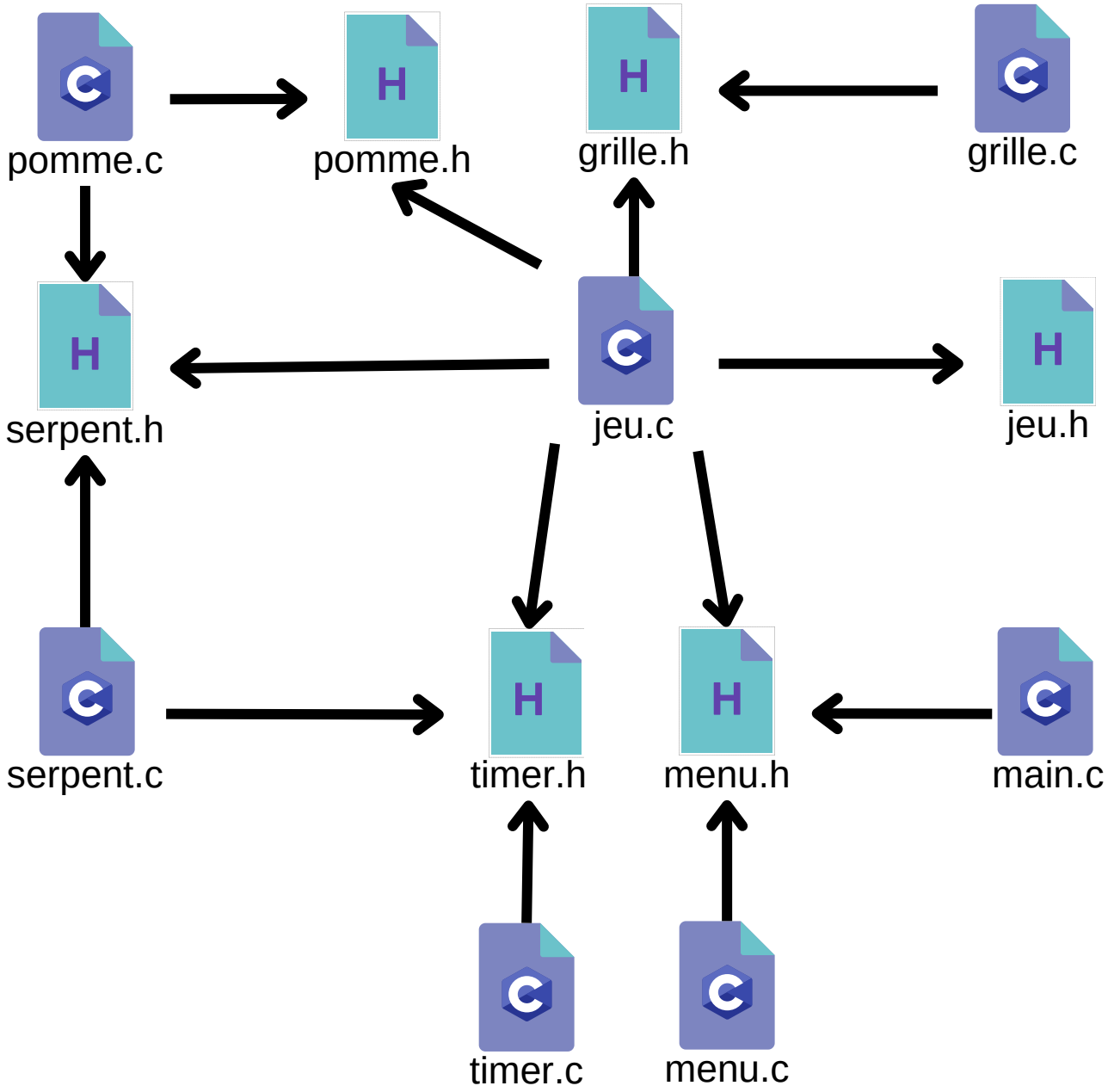
- **But** : Cette fonction a pour objectif de gérer l'affichage des menus en fonction des choix de l'utilisateur. Si l'utilisateur clique sur un bouton spécifique, le programme lance le menu correspondant, et ainsi de suite.

La fonction main

- **But** : La fonction main initialise la fenêtre graphique, gère les différents menus du jeu à l'aide de la fonction menus(), et maintient l'exécution du programme jusqu'à ce que l'utilisateur choisisse de quitter. L'objectif final est de fournir une expérience interactive pour le joueur, lui permettant de naviguer à travers les menus et de sélectionner différents modes de jeu.

MAKEFILE

Voici un schéma représentatif des dépendances des différents fichiers :



PROBLÈMES RENCONTRÉS

Lors de la programmation de notre jeu Snake, Nous avons pu rencontrer plusieurs problèmes :

Dans un premier temps, il a fallu s'adapter à une nouvelle bibliothèque graphique, notamment après avoir été habitués à utiliser Tkinter et Turtle en Python. Cette transition a posé des défis initiaux en termes de familiarisation avec les fonctionnalités, la syntaxe et les particularités propres à la bibliothèque graph.h. Ce changement de cadre a nécessité un temps d'ajustement pour comprendre pleinement les méthodes et les approches spécifiques à cette nouvelle bibliothèque.

Dans un second temps, nous avons été confrontés, tout au long du projet, à une erreur de segmentation, ce qui nous a obligés à recourir à l'utilisation du débogueur pour identifier la cause des problèmes. Ce processus a demandé du temps, mais a été essentiel pour résoudre efficacement ces erreurs et assurer la stabilité du programme.

Pour citer quelques exemples, nous avons tout d'abord eu une première erreur "Segmentation Fault" suite à une mauvaise connaissance de la bibliothèque graphique. En effet, nous avons commis l'erreur maladroite d'incruster dans le code à plusieurs endroit la fonction "FermerGraphique" dans le but de permettre à l'utilisateur de revenir à 0 dans le jeu mais la fonction " FermerGraphique" (issue de la bibliothèque graphique) sert en fait à fermer le mode graphique.

Ensuite, nous avons d'abord essayé de fabriquer un timer à partir de la fonction "Microsecondes" mais un codage maladroit à fait que l'espace mémoire finissait par être très rapidement surchargé et cela provoquait un "Segmentation Fault". nous avons finalement décider de suivre l'exemple généreusement donné dans la documentation de la bibliothèque graphique. Et enfin, une utilisation maladroite de la récursivité nous a causé beaucoup de soucis.

Nous avons aussi pu rencontrer des problèmes avec le serpent tel que nous l'avions initialement codé. En effet, le serpent était simplement défini sous forme d'un simple tableau d'une taille de 2400 (= nombre de cases sur le terrain) et en fin de partie, le serpent n'était pas effacé. Cela a causé, lors des parties suivantes, l'apparition de carrés jaunes fréquemment sur la grille qui pouvais tuer le joueur s'il était au mauvaise endroit au mauvais moment. Pour régler ce problème, nous avons décider de définir le serpent dynamiquement pour ainsi le libéré en fin de partie.

Pour finir, il a été assez difficile de trouver du temps pour le codage en raison d'une surcharge de travail dans d'autres matières (SAé, DS...). Cette contrainte de charge de travail a représenté un défi supplémentaire, nécessitant une gestion efficace de notre temps pour concilier les exigences du projet avec les autres matières du BUT.

NOS AVIS

Moncef STITI : Personnellement, j'ai trouvé ce projet très intéressant car il m'as permis de comprendre comment construire un projet de A à Z. Avant ce projet, il m'était difficile d'imaginer comment mettre en pratique mes connaissances pour créer un projet concret. Maintenant, tout semble beaucoup plus clair et réalisable.
Ce projet a été un vrai défi qui m'a permis d'approfondir mes connaissances en programmation, en particulier dans le domaine des interfaces graphiques (notamment avec la bibliothèque graph.h). La résolution des problèmes/bugs rencontrés pendant le développement a renforcé ma capacité à trouver des solutions efficaces et à collaborer avec un binôme.
De plus, j'ai finis par développer un attachement au fur et à mesure du temps pour se projet, et je compte continuer à rajouter des fonctionnalités avec mon binôme pour améliorer ce jeu.
Pour conclure, malgré une légère surcharge de travaille dans les autres matières, j'ai aimé prendre part à ce projet, et je suis fier du résultat finale que nous avons.

Marco ORFAO : Ce projet de création du jeu Snake m'a beaucoup amusé tout en m'offrant une occasion d'apprendre à mieux utiliser les pointeurs et les structures. La manipulation d'une n-ième BiblioGraphique s'est avérée enrichissante, me faisant prendre conscience de la complexité qu'elle apporte. Cependant, j'ai également réalisé la difficulté d'un travail en groupe séparé, où l'introduction de nouvelles fonctions peut rapidement nous égarer. Toutefois, malgré ces défis, l'expérience globale a contribué de manière significative à mon apprentissage.